AMENDMENTS TO THE SPECIFICATION

1. Please replace paragraph [0037] with the following amended paragraph:

[0037] Moreover, the return address may imply not merely the identity of the calling program, but also a specific location within the calling program. Since the return address is typically the address that immediately follows the call instruction, knowing that the return address is the address of memory location 206(2) may imply that program module 136 was called by a particular function (or even particular line of code) within application program ~~136(2)~~ 135(2).

2. Please replace paragraph [0040] with the following amended paragraph:

[0040] Such an attack on the security of the system can be thwarted by verifying that the caller's code (or at least a portion of the caller's code) has not been modified relative to some known state for that code. Thus, it is desirable to store a checksum, hash, or similar datum that represents the calling code in a known state. For example, in FIG. 4 application 135 is the calling application. Application 135 makes a call to an external program module 136, with a return address 206. Program module 136 regards application 135 as a legitimate caller, and regards calls with return address 206 as coming from a legitimate place within application 135, and this checksum is stored. Checksum 404 may be derived from the entirety of application 135, or from the page(s) that immediately surround return address 206 (reference numeral 402). At the time that program module 136 is called from ~~206~~ application 135, checksum 404 can be recomputed based on application 135 (or pages 402) and then compared with the stored checksum. If the re-computed and stored checksums match, it can be inferred that application 135 (or the relevant pages 402 of application

135) have not been modified relative to a known previous state. Example mechanisms for verifying that a caller has not been modified are more particularly discussed below in the section entitled "module authentication," and in subsequent sections.

3. Please replace paragraph [0058] with the following amended paragraph:

[0058] After the identity of the original caller has been identified based on the return address at 706, the integrity of the caller is verified at 708 ~~706~~ based on a checksum – i.e., a checksum is computed based on the caller (or some portion thereof), and the computed checksum is compared with a stored checksum. Assuming that the caller's integrity is verified, the stack values are set 710 so that the return address points back to the original caller (i.e, the caller who made the call to the intermediate module). A jump is then made 712 into the part of the program module that will perform the function requested (i.e., the function that the original caller desired to perform when making the call at 702), and, when this function is complete, a return instruction is then executed 714 which pops the stack and returns to the return address that was placed on the stack at 710 (which is the same as the return address specified by the original caller).

4. Please replace paragraph [0064] with the following amended paragraph:

[0064] For standard static authentication, the PE (portable executable) file on the persistent storage device (which is usually a disk, but can also be flash memory on a portable device) is hashed and checked against a signed hash value. To compute the file hash, the PE headers of the file must be parsed in order to avoid hashing data that can legitimately change after the file is created. Examples are timestamps and global data sections. The algorithm for hashing the PE file is the same one used by ~~Winows~~WINDOWS™ operating system for signing system binaries.